A Distributed and Parallel Asynchronous Unite and Conquer Method to Solve Large Scale Non-Hermitian Linear Systems

Xinzhe Wu, Serge Petiton

A Distributed and Parallel Asynchronous Unite and Conquer Method to Solve Large Scale Non-Hermitian Linear Systems

Xinzhe WU
Maison de la Simulation, USR 3441
CRIStAL, UMR CNRS 9189
Université de Lille 1, Sciences et Technologies
F-91191, Gif-sur-Yvette, France
xinzhe.wu@ed.univ-lille1.fr

Serge G. Petiton
Maison de la Simulation, USR 3441
CRIStAL, UMR CNRS 9189
Université de Lille 1, Sciences et Technologies
F-91191, Gif-sur-Yvette, France
serge.petiton@univ-lille1.fr

ABSTRACT

Parallel Krylov Subspace Methods are commonly used for solving large-scale sparse linear systems. Facing the development of extreme scale platforms, the minimization of synchronous global communication becomes critical to obtain good efficiency and scalability. This paper highlights a recent development of a hybrid (unite and conquer) method, which combines three computation algorithms together with asynchronous communication to accelerate the resolution of non-Hermitian linear systems and to improve its fault tolerance and reusability. Experimentation shows that our method has an up to 5× speedup and better scalability than the conventional methods for the resolution on hierarchical clusters with hundreds of nodes.

KEYWORDS

linear algebra, iterative methods, asynchronous communication

ACM Reference Format:
Xinzhe WU and Serge G. Petiton. 2018. A Distributed and Parallel Asynchronous Unite and Conquer Method to Solve Large Scale Non-Hermitian Linear Systems. In *HPC Asia 2018: International Conference on High Performance Computing in Asia-Pacific Region, January 28–31, 2018, Chiyoda, Tokyo, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3149457.3154481

1 INTRODUCTION

Scientific applications require the solving of large-scale non-Hermitian linear system $Ax = b$. The collection of Krylov iterative methods, such as the Generalized minimal residual method (GMRES) [20], the Conjugate Gradient (CG) [13] and the Biconjugate Gradient Stabilized Method (BiCGSTAB) [22] are used to solve different kinds of linear systems. These iterative methods approximate the solution x_m of specific matrix A and right-hand vector b from an initial guessed solution x_0. In practice, these methods are always restarted after a specific number of iterations, caused by the augmentation of memory and computational requirements with the increase of the iteration number. These methods are already well implemented in parallel to profit from the great number of computation cores on large clusters. The solving of complicated linear systems with basic iterative methods cannot always converge fast. The convergence rate depends on the specialties of operator matrix. Thus the researchers introduce a kind of preconditioners which combine the stationary methods and iterative methods, to improve the spectral proprieties of operator A and to accelerate the convergence. This kind of preconditioners includes the incomplete LU factorization preconditioner (ILU) [6], the Jacobi preconditioner [5], the successive over-relaxation preconditioner (SOR) [1], etc. Meanwhile, there is a kind of deflated preconditioners which use the approximated eigenvalues during the solving procedure to form a new initial vector for the next restart procedure, which allows to speed up a further computation. Erhel [11] studied a deflated technique for the restarted GMRES algorithm, based on an invariant subspace approximation which is updated at each cycle. Lehoucq [15] introduced a deflation procedure to improve the convergence of an Implicitly Restarted Arnoldi Method (IRAM) for computing the eigenvalues of large matrices. Saad [21] presented a deflated version of the conjugate gradient algorithm for solving linear systems. The implementation of these iterative methods was a good tool to resolve linear systems for a long time during past decades.

Nowadays, the HPC cluster systems continue not only to scale up in compute node and Central Processing Unit (CPU) core count, but also the increase of components heterogeneity with the introduction of the Graphics Processing Unit (GPU) and other accelerators. This trend causes the transition to multi- and many cores inside of computing nodes which communicate explicitly through fast interconnection networks. These hierarchical supercomputers can be seen as the intersection of distributed and parallel computing. Indeed, with a large number of cores, the communication of overall reduction and global synchronization of applications are the bottleneck. When solving a large-scale problem on parallel architectures with preconditioned Krylov methods, the cost per iteration of the method becomes the most significant concern, typically because of communication and synchronization overheads [8]. Consequently, large scalar products, overall synchronization, and other operations involving communication among all cores have to be avoided. The numerical applications should be optimized for more local communication and less global communication. To benefit the full computational power of such hierarchical systems, it is central to explore novel parallel methods and models for the solving of linear systems. These methods should not only accelerate the convergence but also have the abilities to adapt to multi-grain, multi-level

HPC Asia 2018, January 28–31, 2018, Chiyoda, Tokyo, Japan

ACM ISBN 978-1-4503-5372-4/18/01…$15.00
https://doi.org/10.1145/3149457.3154481

memory, to improve the fault tolerance, reduce synchronization and promote asynchronization. The conventional preconditioners have much additional global communication and sparse matrix-vector products. They will lose their advantages on the large-scale hierarchical platforms.

In order to explore the novel methods for modern computer architectures, Emad [9] proposed the unite and conquer approach. This approach is a model for the design of numerical methods by combining different computation components together to work for the same objective, with asynchronous communication among them. Unite implies the combination of different calculation components, and conquer represents different components work together to solve one problem. In the unite and conquer methods, different computation parallel components work independently with asynchronous communication. These different components can be deployed on different platforms such as P2P, cloud and the supercomputer systems, or on the same platform with different processors. The idea of unite and conquer approach came from the article of Saad [17] in 1984, where he suggested using Chebyshev polynomial to accelerate the convergence of Explicitly Restarted Arnoldi Method (ERAM) to solve eigenvalue problems. Brezinski [4] proposed in 1994 an approach for solving a system of linear equations which takes a combination of two arbitrary approximate solutions of two methods. In 2005, Emad [10] proposed a hybrid approach based on a combination of multiple ERAMs, which showed significant improvement in solving different eigenvalue problems. In 2016, Fender [12] studied a variant of multiple IRAMs and generated multiple subspaces in a nested fashion in order to dynamically pick the best one inside each restart cycle.

Inspired by the unite and conquer approach, this paper introduces a recent development of unite and conquer method to solve large-scale non-Hermitian sparse linear systems. This method comprises three computation components: ERAM Component, GMRES Component and LS (Least Squares) Component. GMRES Component is used to solve the systems, LS Component and ERAM Component serve as the preconditioning part. The key feature of this hybrid method is the asynchronous communication among these three components, which reduces the number of overall synchronization points and minimizes the global communication. This method is called Unite and Conquer GMRES/LS-ERAM (UCGLE) method.

There are three levels of parallelisms in UCGLE method to explore the hierarchical computing architectures. The convergence acceleration of UCGLE method is similar with a deflated preconditioner. The difference between them is that the improvement of the former one is intrinsic to the methods. It means that in the deflated preconditioning methods, for each time of preconditioning, the solving procedure should stop and wait for the temporary preconditioning procedure. Asynchronous communication of the latter can cover the synchronous communication overhead.

Obviously, the asynchronous communication among the different computation components improves the fault tolerance and the reusability of this method. The three computation components work independently from each other, when errors occur inside of ERAM Component, GMRES Component or LS Component, UCGLE can continue to work as a normal restarted GMRES method to solve the problems. In fact, the materials for accelerating the convergence are the eigenvalues. With the help of asynchronous communication, we can select to save the computed eigenvalues by ERAM method into a local file and reuse it for the other solving procedures with the same matrix.

We implement UCGLE method based on the scientific libraries PETSc and SLEPc for both CPU and GPU versions. We make use of these mature libraries in order to focus on the prototype of the asynchronous model instead of exploiting the optimization of codes performance inside of each component. PETSc provides also different kinds of preconditioners which can be easily used to the performance comparison with UCGLE method.

In Section 2, we present the three basic numerical algorithms in detail which construct the computation components of UCGLE method. The implementation of different levels parallelism and communication are shown in Section 3. In Section 4, we evaluate the convergence and performance of UCGLE method with our scientific large-scale sparse matrices on top of hierarchical CPU/GPU clusters. We give the conclusions and perspectives in Section 5.

2 COMPONENTS NUMERICAL ALGORITHMS

In linear algebra, the m-order Krylov subspace generated by a $n \times n$ matrix A and a vector b of dimension n is the linear subspace spanned by the images of b under the first m powers of A, that is

$$K_m(A, b) = span(b, Ab, A^2b, \cdots, A^{m-1}b)$$

The Krylov iterative methods are often used to solve large-scale linear systems and eigenvalue problems. In this section, we present in detail the three basic numerical algorithms used by UCGLE.

2.1 ERAM Algorithm

Algorithm 1 Arnoldi Reduction

```
1: function AR(input:A, m, v, output: H_m, Ω_m)
2:     ω_1 = v/||v||_2
3:     for j = 1, 2, ··· , m − 1 do
4:         h_{i,j} = (Aω_j, ω_i), for i = 1, 2, ··· , j
5:         ω_j = Aω_j − Σ_{i=1}^{j} h_{i,j} ω_i
6:         h_{j+1,j} = ||ω_j||_2
7:         ω_{j+1} = ω_j / h_{j+1,j}
8:     end for
9: end function
```

Arnoldi algorithm is a well-known method to approximate the eigenvalues of large sparse matrices, which was firstly proposed by W. E. Arnoldi in 1951 [2]. The kernel of Arnoldi algorithm is the Arnoldi reduction, which gives an orthonormal basis $\Omega_m = (\omega_1, \omega_2, \cdots, \omega_m)$ of Krylov subspace $K_m(A, v)$, by the Gram-Schmidt orthogonalization, where A is $n \times n$ matrix, and v is a n-dimensional vector. Arnoldi reduction can transfer a matrix A to be an upper Hessenberg matrix H_m, the eigenvalues of H_m are the approximated ones of A, which are called the Ritz values of A. See Algorithm 1 for the Arnoldi reduction in detail. With the Arnoldi reduction, the r desired Ritz values $\Lambda_r = (\lambda_1, \lambda_2, \cdots, \lambda_r)$, and the corresponding Ritz vectors $U_r = (u_1, u_2, \cdots, u_r)$ can be calculated by Basic Arnoldi method.

The numerical accuracy of the computed eigenpairs of basic Arnoldi method depends highly on the size of the Krylov subspace and the orthogonality of Ω_m. Generally, the larger the subspace is, the better the eigenpairs approximation is. The problem is that firstly the orthogonality of the computed Ω_m tends to degrade with each basis extension. Also, the larger the subspace size is, the larger the Ω_m matrix gets. Hence available memory may also limit the subspace size, and so the achievable accuracy of the Arnoldi process. To overcome this, Saad [19] proposed to restart the Arnoldi process, which is the ERAM. ERAM is an iterative method whose main core is the Arnoldi process. The subspace size is fixed as m, and only the starting vector will vary. After one restart of the Arnoldi process, the starting vector will be initialized by using information from the computed Ritz vectors. In this way, the vector will be forced to be in the desired invariant subspace. The Arnoldi process and this iterative scheme will be executed until a satisfactory solution is computed. The Algorithm of ERAM is given by Algorithm 2, where ϵ_a is a tolerance value, r is desired eigenvalues number and the function g defines the stopping criterion of iterations.

Algorithm 2 Explicitly Restarted Arnoldi Method

1: **function** ERAM(*input*: A, r, m, v, ϵ_a, *output*: Λ_r)
2: Compute an AR(*input*:A, m, v, *output*: H_m, Ω_m)
3: Compute r desired eigenvalues λ_i ($i \in [1, r]$) of H_m
4: Set $u_i = \Omega_m y_i$, for $i = 1, 2, \cdots, r$, the Ritz vectors
5: Compute $R_r = (\rho_1 . \cdots, \rho_r)$ with $\rho_i = ||\lambda_i u_i - Au_i||_2$
6: **if** $g(\rho_i) < \epsilon_a$ ($i \in [1, r]$) **then**
7: stop
8: **else**
9: set $v = \sum_{i=1}^{d} Re(v_i)$, and GOTO 2
10: **end if**
11: **end function**

2.2 GMRES Algorithm

GMRES is a Krylov iterative method to solve non-Hermitian linear systems. It approximates the solution x_m of matrix A and right hand vector b from an initial guessed solution x_0, with the minimal residual in a Krylov subspace $K_m(A, v)$, which is given by Algorithm 3. The GMRES method was introduced by Youssef Saad and Martin H. Schultz in 1986 [20].

Algorithm 3 Basic GMRES method

1: **function** BASICGMRES(*input*: A, m, x_0, b, *output*: x_m)
2: $r_0 = b - Ax_0, \beta = ||r_0||_2$, and $v_1 = r_0/\beta$
3: Compute an AR(*input*:A, m, v_1, *output*: H_m, Ω_m)
4: Compute y_m which minimizes $||\beta e_1 - H_m y||_2$
5: $x_m = x_0 + \Omega_m y_m$
6: **end function**

If GMRES method is restarted after a number of iterations, to avoid enormous memory and computational requirements with the increase of Krylov subspace projection number. It is called the restarted GMRES. The restarted GMRES won't stop until the condition $||b - Ax_m|| < \epsilon_g$ is satisfied. See Algorithm 4 for restarted GMRES algorithm in detail. A well-known difficulty with the restarted GMRES algorithm is that it can stagnate when the matrix is not positive definite. A typical method is to use preconditioning techniques whose goal is to reduce the number of steps required to converge.

Algorithm 4 Restarted GMRES method

1: **function** RESTARTEDGMRES(*input*: A, m, x_0, b, ϵ_g, *output*: x_m)
2: BASICGMRES(*input*: A, m, x_0, b, *output*: x_m)
3: **if** ($||b - Ax_m|| < \epsilon_g$) **then**
4: Stop
5: **else**
6: set $x_0 = x_m$ and GOTO 2
7: **end if**
8: **end function**

2.3 Least Square Polynomial Algorithm

The Least Squares polynomial method is a kind of iterative methods to solve linear systems, which aims to calculate a new preconditioned residual for restarted GMRES in the UCGLE method. The iterates of the Least Squares method can be written as $x_n = x_0 + P_n(A)r_0$, where x_0 is a selected initial approximation to the solution, r_0 the corresponding residual norm, and P_n a polynomial of degree $n-1$. We set a polynomial of n degree R_n such that

$$R_n(\lambda) = 1 - \lambda P_n(\lambda)$$

.

The residual of n^{th} steps iteration r_n can be expressed as equation $r_n = R_n(A)r_0$, with the constraint $R_n(0) = 1$. We want to find a kind of polynomial which can minimize $||R_n(A)r_0||_2$, with $||.||_2$ the Euclidean norm.

If A is a diagonalizable matrix with its spectrum denoted as $\sigma(A) = \lambda_1, \cdots, \lambda_n$, and the associated eigenvectors $u_1, \cdots, u_n$. Expanding the residual vector r_n in the basis of these eigenvectors as as $r_n = \sum_{i=1}^{n} R_n(\lambda_i)\rho_i u_i$, which allows to get the upper limit of $||r_n||$ as

$$||r_0||_2 \max_{\lambda \in \sigma(A)} |R_n(\lambda)| \quad (1)$$

In order to minimize the norm of r_n, it is possible to find a polynomial P_n which can minimize the Equation (1).

In article [16], Manteuffel proposed to expand P_n with a basis of Chebyshev polynomial $t_j(\lambda) = \frac{T_j \frac{\lambda - c}{d}}{T_j \frac{c}{d}}$, where t_i is constructed by an ellipse englobing the convex hull using the computed eigenvalues, with c the centre of ellipse, and d the focal distance of ellipse. P_n is under form that $P_n = \sum_{i=0}^{n-1} \eta_i t_i$. The selected Chebyshev polynomials t_i meet the three terms recurrence relation (2).

$$t_{i+1}(\lambda) = \frac{1}{\beta_{i+1}}[\lambda t_i(\lambda) - \alpha_i t_i(\lambda) - \delta_i t_{i-1}] \quad (2)$$

For the computation of parameters $H = (\eta_0, \eta_1, \cdots, \eta_{n-1})$, we construct a modified gram matrix M_n with dimension $n \times n$, and matrix T_n with dimension $(n+1) \times n$ by the three terms recurrence of the basis t_i. M_n can be factorized to be $M_n = LL^T$ by the Cholesky

factorization. The parameters H can be computed by a least squares problem of the formula

$$min\|l_{11}e_1 - F_dH\| \tag{3}$$

With the definition of vectors $\omega_i \in IR^n$ by $\omega_i = t_i(A)r_0$, we can obtain the Equation (4), and in the end iteration (5).

$$\omega_{i+1} = \frac{1}{\beta_{i+1}}(A\omega_i - \alpha_i\omega_i - \delta_i\omega_{i-1}) \tag{4}$$

$$x_n = x_0 + P_n(A)r_0 = x_0 + \sum_{i=1}^{n-1}\eta_i\omega_i \tag{5}$$

The pseudocode of this method is presented in Algorithm 5, where A is a $n \times n$ matrix, b is a right-hand vector of dimension n, d is the degree of Least Squares polynomial, Λ_r the collection of approximate eigenvalues, and the output values are $A_d = (\alpha_0, \alpha_1, \cdots, \alpha_{d-1})$, $B_d = (\beta_1, \beta_2, \cdots, \beta_d)$, $\Delta_d = (\delta_1, \delta_2, \cdots, \delta_{d-1})$, and $H_d = (\eta_0, \eta_1, \cdots, \eta_{d-1})$, which will be used for constitution of a new GMRES initial vector. a, c, d are the required parameters to fix an ellipse in the plan, with a the distance between the vertex and centre, c the centre position and d the focal distance. For more details of Least Squares iterative method, see[18].

Algorithm 5 Least Square method

```
1:  function LS(input: A, b, d, Λ_r, output: A_d, B_d, Δ_d, H)
2:      construct the convex hull C by Λ_r
3:      construct ellispe(a, c, d) by the convex hull C
4:      compute parameters A_d, B_d, Δ_d by ellispe(a, c, d)
5:      construct matrix T (d + 1) × d matrix by A_d, B_d, Δ_d
6:      construct Gram matrix M_d by Chebyshev polynomials basis
7:      Cholesky factorization M_d = LL^T
8:      F_d = L^T T
9:      H_d satisfies min ||l_11 e_1 − F_d H||
10: end function
```

3 UCGLE METHOD IMPLEMENTATION

3.1 Workflow and Parameters Analysis

UCGLE method comprises mainly two parts: the first part uses the restarted GMRES method to solve the linear systems; in the second part, it computes a specific number of approximated eigenvalues, and then applies them to the Least Squares method and gets a new preconditioned residual, as a new initial vector for restarted GMRES. Suppose that the computed convex hull by Least Squares contains eigenvalues $\lambda_1, \cdots, \lambda_m$, the residual given by Least Square is

$$r = \sum_{i=1}^{m}\rho((R_k)(\lambda_i)^\iota)u_i + \sum_{i=m+1}^{n}\rho((R_k)(\lambda_i)^\iota)u_i$$

The first part of this residual is small as the Least Squares method finds R_k minimizing $|R_k(\lambda)|$ in the convex hull, but not with the second part, where the residual will be rich in the eigenvectors associated with the eigenvalues outside the convex hull. With the number of approximated eigenvalues ι increasing, the first part will be much closer to zero and the second part will be much larger. This results in an enormous increase of restarted GMRES preconditioned vector norm. Meanwhile, when restarted GMRES restarts with the combination of a number of eigenvectors, the convergence will be faster even if the residual is enormous.

Figure 1 gives the workflow of UCGLE method with three computation components. ERAM Component and GMRES Component are implemented in parallel, and the communication among them is asynchronous. ERAM Component computes a desired number of eigenvalues, and then sends them to LS Component; LS Component uses these received eigenvalues to output a new residual vector, and sends it to GMRES Component; GMRES Component uses this residual as a new restarted initial vector for solving non-Hermitian linear systems.

UCGLE method is a combination of three different methods, there are a number of parameters, which have impacts on its convergence rate. We summarize these different related ones, and classify them according to their relations with different components.

I. GMRES Component
   * m_g: GMRES Krylov Subspace size
   * ϵ_g: absolute tolerance for the GMRES convergence test
   * P_g: GMRES core number
   * s_{use}: number of times that polynomial applied on the residual before taking account into the new eigenvalues
   * L: number of GMRES restarts between two times of LS preconditioning

II. ERAM Component
   * m_a: ERAM Krylov subspace size
   * r: number of eigenvalues required
   * ϵ_a: tolerance for the ERAM convergence test
   * P_a: ERAM core number

III. LS Component
   * d: Least Squares polynomial degree

The Algorithm 6 shows the implementation of UCGLE's three components and their asynchronous communication in detail. ERAM Component loads the parameters m_a, v, r, ϵ_a and the operator matrix A, then launches ERAM function. When it receives a new vector X_TMP from GMRES Component, this vector will be stored in ERAM Component. This vector is updated with the continuous receiving of a new one from GMRES Component. If the r eigenvalues Λ_r are approximated by ERAM Component, it will send them to LS Component, at the same time, it is able to save the eigenvalues into the local file.

LS Component won't start work until it receives the eigenvalues Λ_r sent from ERAM Component. Then it will use them to compute the parameters A_d, B_d, Δ_d, H_d, whose dimensions are related to LS parameter d, the Least Squares polynomial degree, and send these parameters to GMRES Component.

GMRES Component loads the parameters $A, m_g, x_0, b, \epsilon_g, L, s_{use}$ to solve the linear systems. At the beginning of the execution, it behaves like the basic GMRES method. When it finishes the m^{th} iteration, it will check if the condition $||b - Ax_m|| < \epsilon_g$ is satisfied, if yes, x_m is the solution of linear system $Ax = b$, or GMRES Component will be restarted using x_m as a new initial vector. A parameter *count* is used to count the times of restart. All these processes are similar as a Restarted GMRES. But when *count* is an integer multiple of L (number of GMRES restarts between two times preconditioning of LS), it will check if it has received the parameters

A_d, B_d, Δ_d, H_d from LS Component. If yes, these parameters will be used to construct a preconditioning polynomial P_d, which can be used to generate a preconditioned residual x_d, then set the initial vector x_0 as x_d, and restart the basic GMRES, until the exit condition is satisfied.

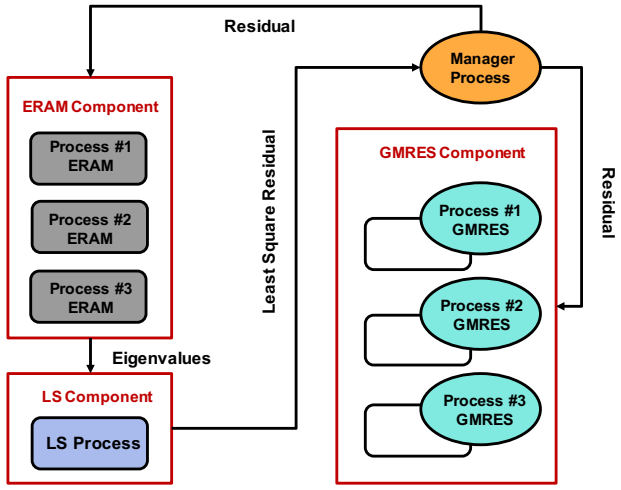

Figure 1: Workflow of UCGLE method's three components

3.2 Distributed and Parallel Communications

UCGLE method is a distributed parallel method which can profit both shared memory and distributed memory of computational architectures. As shown in Figure 2, it has two levels of parallelism for distributed memory: 1) Coarse Grain/Component level: UCGLE allows the distribution of different numerical components, including the preconditioning part (LS and ERAM) and the solving part (GMRES) on different platforms or processors; 2) Medium Grain/Intra-component level, GMRES and ERAM components are both deployed in parallel; the third level for shared memory is the Fine Grain/Thread parallelism: the OpenMP thread level parallelism in CPU, or the accelerator level parallelism if GPUs or other accelerators are available.

The GMRES method has been implemented by PETSc, and the ERAM method is provided by SLEPc. Additional functions have been added to the GMRES and ERAM provided by PETSc and SLEPc in order to include the sending and receiving functions of different types of data. For the implementation of LS Component, it computes the convex hull and the ellipse encircling the Ritz values of matrix A, which allows generating a novel Gram matrix M of selected Chebyshev polynomial basis. This matrix should be factorized into LL^T by the Cholesky algorithm. The Cholesky method is ensured by PETSc as a preconditioner but can be used as a factorization method. The implementation based on these libraries allows the recompilation of the UCGLE codes to adapt into both CPU and GPU architectures. The experimentation of this paper does not consider the OpenMP thread level of parallelism since the implementation of PETSc and SLEPc is not thread-safe due to their complicated data structures. The data structures of PETSc and SLEPc makes it more

Algorithm 6 Implementation of Components

function LOADERAM(*input*: A, m_a, v, r, ϵ_a)
  while exit==False **do**
    ERAM(A, r, m_a, v, ϵ_a, *output*: Λ_r)
    Send (Λ_r) to LS
    if $saveflg == TRUE$ **then**
      write (Λ_r) to file *eigenvalues.bin*
    end if
    if Recv (X_TMP) **then**
      update X_TMP
    end if
    if Recv ($exit == TRUE$) **then**
      Send (*exit*) to LS Component and stop
    end if
  end while
end function
function LOADLS(*input*: A, b, d)
  if Recv(Λ_r) **then**
    LS(*input*: A, b, d, Λ_r, *output*: A_d, B_d, Δ_d, H_d)
    Send (A_d, B_d, Δ_d, H_d) to GMRES Component
  end if
  if Recv ($exit == TRUE$) **then**
    stop
  end if
end function
function LOADGMRES(*input*: $A, m_g, x_0, b, \epsilon_g, L, s_{use}$, *output*: x_m)
  $count = 0$
  BASICGMRES(*input*: A, m, x_0, b, *output*: x_m)
  $X\_TMP = x_m$
  Send (X_TMP) to ERAM Component
  if $||b - Ax_m|| < \epsilon_g$ **then**
    return x_m
    Send ($exit == TRUE$) to ERAM Component and stop
  else
    if $count \mid L$ **then**
      if recv (A_d, B_d, Δ_d, H_d) **then**
        $r_0 = f - Ax_0$, $\omega_1 = r_0$ and $x_0 = 0$
        for $k = 1, 2, \cdots, s_{use}$ **do**
          for $i = 1, 2, \cdots, d-1$ **do**
            $\omega_{i+1} = \frac{1}{\beta_{i+1}}[A\omega_i - \alpha_i\omega_i - \delta_i\omega_{i-1}]$
            $x_{i+1} = x_i + \eta_{i+1}\omega_{i+1}$
          end for
        end for
        set $x_0 = x_d$, and GOTO 1
        $count++$
      end if
    else
      set $x_0 = x_m$, and GOTO 1
      $count++$
    end if
  end if
end function

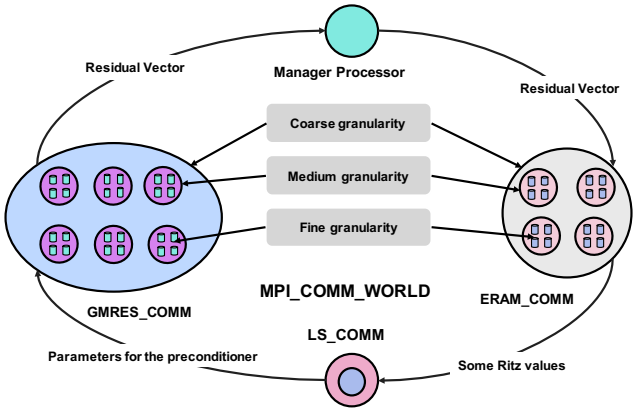

Figure 2: Communication and different levels parallelism of UCGLE method

difficult to partition the data among the threads to prevent conflict and to achieve good performance [3].

The main characteristic of UCGLE method is its asynchronous communication. But the synchronous communication takes place inside of GMRES and ERAM components. Distributed and parallel communication involves different types of exchange data, such as vectors, scalar tables, and signals among different components. When the data are sent and received in a distributed way, it is essential to ensure the consistency of data. In our case, we choose to introduce an intermediate node as a proxy to carry out only several types of exchanges, and thus facilitate the implementation of asynchronous communication. This proxy is called *Manager Process* as in Figure 2. One process can fulfill all the data exchanges.

Asynchronous communication allows each computation component to conduct independently the work assigned to it without waiting for the input data. The asynchronous data sending and receiving operations are implemented by the non-blocking communication of Message Passing Interface (MPI). Sending takes place after the sender has completed the task assigned to it. Before any prior shipment, the component checks whether several transactions are now on the way. If yes, this task will be canceled to avoid the competition of different types of sending tasks. Sent data are copied into a buffer to prevent them from being modified while sending. For the asynchronous data receiving, before starting this task, the component will check if data is expected to be received. Once the receiving buffer is allocated, the component performs the receiving of data while respecting the distribution of data globally according to the rank of sending processes. It is also important to validate the consistency of receiving data before any use of them by the tasks assigned to the components.

From a view of asynchronous communication, the implementation of UCGLE is to establish of several communicators inside of *MPI_COMM_WORLD* and their inter-communication. The topology of communication among these groups is a circle shown in Figure 2. The total number of computing units supplied by the user is thus divided into four groups according to the following distribution: P_t is the total number of processes, then $P_t = P_g + P_a + P_l + P_m$, where P_g is the number of processes assigned to GMRES Component, P_a the number of processes to ERAM component, P_l the number of processes allocated to LS Component and P_f the number of processes allocated to *ManagerProcess* proxy. P_g and P_a are greater than or equal to 1, P_l and P_m are both exactly equal to 1. LS Component is a serial component because the Least Squares polynomial method cannot be parallelized.

P_t is thus divided into several MPI groups according to a color code. The minimum number of processes that our program requires is 4. We utilize the mechanism of MPI standard to fully support the communication of our application. The communication layer that does not depend on the application, this allows the replacement and scalability of various components provided.

3.3 Reusability Analysis

In this section, we study the potential reusability of UCGLE method, which is assured by its asynchronous communication. Indeed, the eigenvalues are used to improve the convergence rate of linear systems by GMRES method. These eigenvalues approximated by ERAM Component can be saved into a local file. For the next time of a different linear system with the same operator matrix, these eigenvalues can be directly reloaded from the local file by LS Component and execute the preconditioning procedure. This reusability proposes also a new strategy of resolving a series of linear systems in sequence with the same matrix and different right-hand sides. This type of multiple resolving linear systems is well needed in various scientific fields. This strategy is similar to a traditional GMRES method using the Least Squares polynomial method as a deflated preconditioner. But the LS Component and GMRES Component communication keeps asynchronous, thus the preconditioning on the restarted GMRES can be flexible, we can control the frequency of preconditioning and restart (the parameter L). In the experiments, we can propose an autotuning strategy to get an optimized L for specific linear systems.

It seems if we compute a specific number of eigenvalues before the first time computation and then load them for all the resolving procedures with the same matrix, ERAM component will be not needed, and the existence of UCGLE method will be questioned. In fact, the speedup of UCGLE method depends on the quality and quantity of approximated eigenvalues which cannot always be quickly approximated. The more eigenvalues are calculated, the more accurate these eigenvalues are, the more significant the acceleration of LS preconditioning will be. The multiple solving different linear systems with the same matrix by UCGLE allows the augmentation of eigenvalues number and the amelioration of these values. The reusability of UCGLE method will be presented in future as the page limitation of this article.

3.4 Fault Tolerance Analysis

One important property of the asynchronous UCGLE algorithm is its fault tolerance. That means, the loss of either GMRES Component or ERAM Component at run time doesn't impede the whole computation.

To be more precise:

1) If ERAM computing units are in fault, GMRES Component can continue to run as a classic GMRES method without receiving

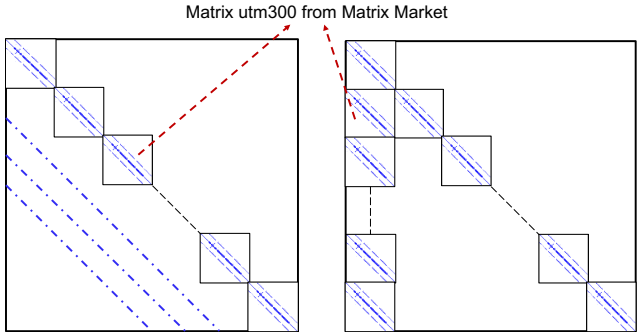

Figure 3: Two strategies of large and sparse matrix generator by a original matrix utm300 of Matrix Market

the eigenvalues from ERAM Component and the acceleration of LS algorithm.

2) If GMRES computing units are in fault, the fault tolerance mechanism will be more complex. In this situation, firstly the tasks of ERAM Component will be canceled, secondly, these released computing units will be reset as a GMRES Component to continue the resolving procedure without acceleration. The feasibility of replacing ERAM by GMRES is guaranteed. The required materials to retake the resolving task is the operator matrix A, the right-hand side b and the temporary solution x_m. The two former ones have been loaded along with the set-up of UCGLE method. The x_m is ensured to be on the former ERAM computing units since it can be sent and received by the asynchronous communication between GMRES and ERAM Components.

The simulation of ERAM and GMRES Components' fault tolerance will be given in Section 4.5.

4 EVALUATION

In this section, we evaluate the acceleration of convergence and the scaling performance of UCGLE method comparing with selected classic preconditioners using the four selected large-scale matrices on both CPU and GPU platforms.

4.1 Hardwares

In experiments, we implement UCGLE method on a cluster ROMEO. ROMEO is located at University of Reims Champagne-Ardenne of France. It is a heterogeneous system made of Xeon CPUs and Nvidia GPUs, with 130 BullX R421 nodes, each node composes 2 processors Intel Ivy Bridge 8 cores @ 2.6 GHz, 2 NVIDIA Tesla K20X accelerators, and 32 GB DDR memory. The exact information of ROMEO is given in Table 1.

4.2 Test Sparse Matrices

UCGLE method has been tested with different matrices, both industrial and generated. Our purpose is to test this algorithm on large sparse linear systems. We have successfully evaluated UCGLE with a number of sparse matrices from Matrix Market. However, these matrices are small compared to the desired sizes. Thus we proposed a matrix generator to create several large-scale linear systems. Additionally, the speedup of UCGLE method depends on the spectrum

Table 1: Node Specifications of the cluster ROMEO

Nodes Number	BullX R421 × 130
Mother Board	SuperMicro X9DRG-QF
CPU	Intel Ivy Bridge 8 cores 2,6 GHz × 2 sockets
Memory	DDR 32GB
GPU	NVIDIA Tesla K20X × 2
Memory	GDDR5 6 GB / GPU

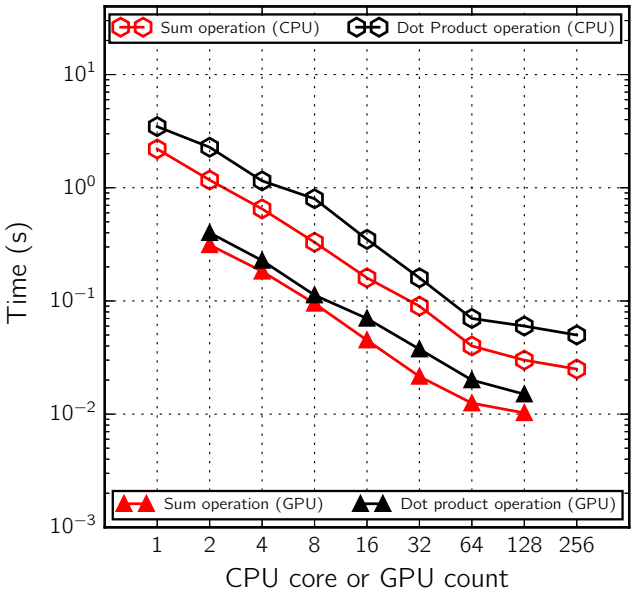

Figure 4: Global synchronous communication evaluation by parallel sum and dot product operations on ROMEO; X-axis refers respectively to the CPU core number from 1 to 256 and the GPU core number from 2 to 128; Y-axis refers to the operation time; a base 10 logarithmic scale is used for Y-axis and a base 2 logarithmic scale is used for X-axis.

of linear systems. We have used our scientific large-scale matrices with known eigenvalues to test the UCGLE method.

4.2.1 Matrix Generation. We developed this parallel sparse matrix generator based on MPI and PETSc, which reads an industrial matrix of Matrix Market collection as an initial one to build larger ones. This generator allows building a new matrix A by performing several copies of a same small unsymmetrical matrix B onto the diagonal. In order to keep the generated matrices being unsymmetrical and especially non-block in diagonal, we propose two different strategies to add the values on the off-diagonal of A, as shown in the Figure 3, the first one is called ML type matrix, and the second is called MB type matrix. The reason of adding different values on the off-diagonal is to ensure that the eigenvalues of newly generated matrix won't be the same as the original one, and the convergence rate won't be too fast.

For the generation of matrix ML, several parallel lines with different values can be added to the off-diagonal. The good selection

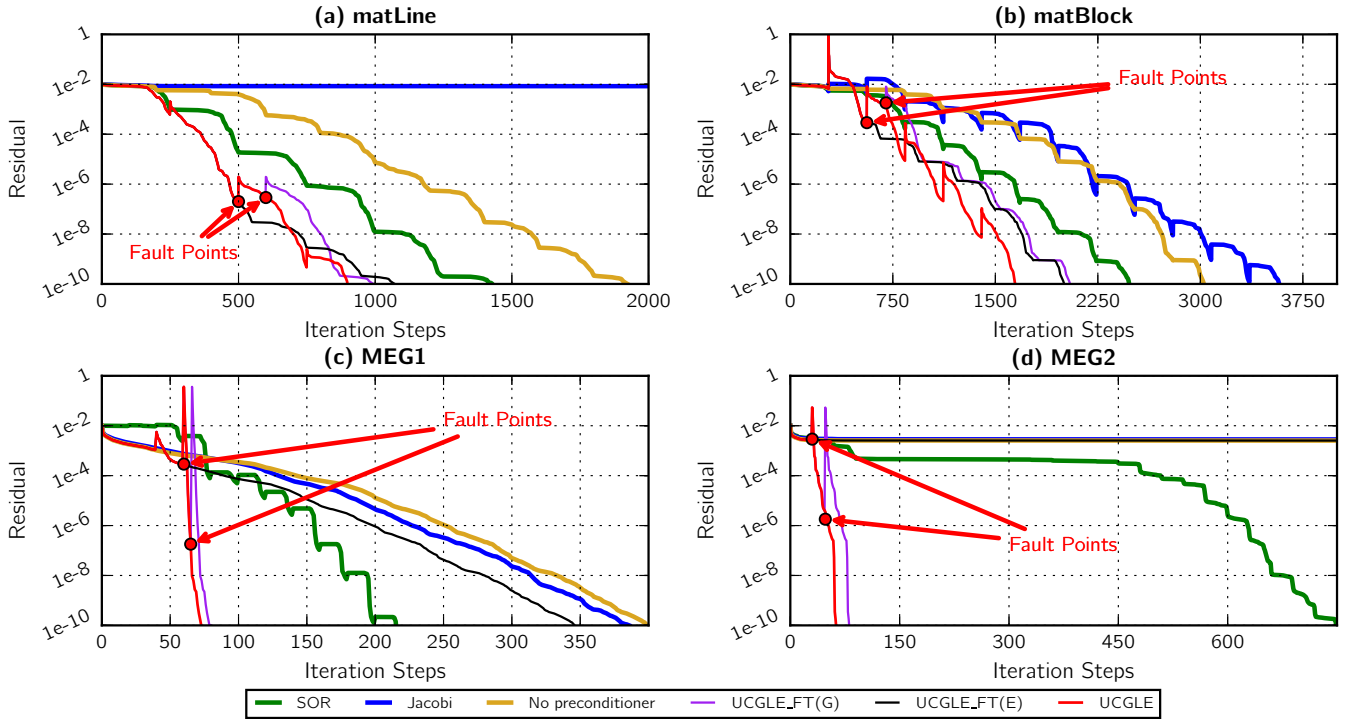

Figure 5: Convergence comparison of *matLine*, *matBlock*, *MEG*1 and *MEG*2 by UCGLE, classic GMRES, Jacobi preconditoned GMRES, SOR preconditioned GMRES, UCGLE_FT(G) and UCGLE_FT(E); X-axis refers to the iteration step for each method; Y-axis refers to the residual, a base 10 logarithmic scale is used for Y-axis; GMRES restarted parameters for *matLine*, *matBlock*, *MEG*1, *MEG*2 are respectively 250, 280, 30 and 40; ERAM fault points are respectively 500, 560, 60,30, and GMRES fault points are 600, 700, 70 and 48.

Table 2: Test matrices information

Matrix Name	n	nnz	Matrix Type
matLine	1.8×10^7	2.9×10^7	non-Symmetric
matBlock	1.8×10^7	1.9×10^8	non-Symmetric
*MEG*1	1.024×10^7	7.27×10^9	non-Hermitian
*MEG*2	5.1×10^6	3.64×10^9	non-Hermitian

of added values can prevent the generated matrix to converge fast with the basic iterative solvers. The way to generate the *MB* type matrix is much easier, the original matrix is copied on the diagonal and the first block column matrix as shown in the Figure 3.

4.2.2 Test matrices. We have selected four different matrices to evaluate UCGLE method. The matrix *matLine* is a *ML* type matrix, and *matBlock* is a *MB* type matrix, they are both generated by the industrial matrix *utm*300 which can be downloaded from the Matrix Market. The distribution of eigenvalues in the complex plane has an important impact on the convergence of linear systems. We have selected two scientific matrices with known eigenvalues: *MEG*1 and *MEG*2. The eigenvalues of the matrix *MEG*1 and *MEG*2 have different eigenvalues distribution in the complex plane. The Table 2 gives the details of these test matrices.

4.3 Global Communication Evaluation

In the parallel algorithms, often one must synchronize the communication. The parallel computation of dot product and the sum reduction are the good examples. Synchronization is needed for the execution of these operations. Before the test of UCGLE method, we evaluate the global synchronous communication of both homogeneous and heterogeneous architecture platforms by the parallel operations of sum and dot product with a real array of size 1.0×10^9. In Figure 4, we can conclude that these global reductions will lose its good scalability if the computing unit number is larger than 64, although ROMEO platform has only hundreds of cores. It can be predicted that this situation will be much worse on the coming exascale platforms. In this background, UCGLE is proposed to promote local communication and reduce global synchronous communication.

4.4 Convergence Evaluation

We evaluate the convergence acceleration of four large-scale matrices *matLine*, *matBlock*, *MEG*1, *MEG*2 using different methods: 1) UCGLE, 2) restarted GMRES without preconditioning, 3) restarted GMRES with SOR preconditioner, 4) restarted GMRES with Jacobi preconditioner. We select the Jacobi and SOR preconditioners for the experimentations because they two are well implemented in

Table 3: Summary of iteration number for convergence of 4 test matrices using SOR, Jacobi, non preconditioned GMRES,UCGLE_FT(G),UCGLE_FT(G) and UCGLE: red × in the table presents this solving procedure cannot converge to accurate solution (here absolute residual tolerance 1×10^{-10} for GMRES convergence test) in acceptable iteration number (20000 here).

Matrix Name	SOR	Jacobi	No preconditioner	UCGLE_FT(G)	UCGLE_FT(G)	UCGLE
$matLine$	1430	×	1924	995	1073	900
$matBlock$	2481	3579	3027	2048	2005	1646
$MEG1$	217	386	400	81	347	74
$MEG2$	750	×	×	82	×	64

parallel by PETSc. The GMRES restarted parameter for *matLine*, *matBlock*, *MEG*1, *MEG*2 are respectively 250, 280, 30 and 40.

Figure 5 compares the convergence curves of experimentation, and the Table 3 gives the convergence steps of each method with 4 test matrices in details. We find that UCGLE method has spectacular acceleration on the convergence rate comparing these conventional preconditioners. It has almost two times of acceleration for *matLine*, *matBlock* and *MEG*1 matrices, and more than 10 times of acceleration for *MEG*2 than the conventional preconditioner SOR. The SOR preconditioner is already much better than the Jacobi preconditioner for the test matrices.

4.5 Fault Tolerance Evaluation

The fault tolerance of UCGLE method is also studied by the simulation of loss of either GMRES or ERAM Components. UCGLE_FT(G) in Figure 5 represents the fault tolerance simulation of GMRES Component, and UCGLE_FT(E) implies the fault tolerance simulation of ERAM Component.

The failure of ERAM Component is simulated by fixing the execution loop number of ERAM algorithm, in this case, ERAM exits after a fixed number of solving procedures. We mark the ERAM fault points of four matrices in Figure 5: respectively 500, 560, 60 and 30 iteration step for each case. The UCGLE_FT(E) curves of four experimentations show that GMRES Component will continue to resolve the systems without LS acceleration. The Table 3 shows that the iteration number is greater than the normal UCGLE method but less than the GMRES method without preconditioning.

The failure of GMRES Component is simulated by setting the allowed iteration number of GMRES algorithm to be much smaller than the needed iteration number for convergence. The values of these four cases are respectively 600, 700, 70 and 48. They are also marked in Figure 5. In this figure, after the quitting of GMRES Component without the finish of its task, ERAM computing units will automatically take over the position of GMRES component. The new GMRES resolving procedure will use the temporary solution x_m as a new restarted initial vector received asynchronously from the previous restart procedure of GMRES Component before its failure. In this case, ERAM Component no longer exists, thus the resolving task can be continued as the classic GMRES without preconditioning. In Figure 5, the UCGLE_FT(E) curves of four experimentations give the simulation of this case. We can find there's the difference between UCGLE_FT(E) and UCGLE_FT(G). In UCGLE_FT(G), the new GMRES Component takes x_m of previous restart procedure, thus it will repeat the iteration steps of previous restart iterations until the failure of GMRES. Another fact of UCGLE_FT(G) which cannot be concluded from Figure 5, but can be easily obtained, is that the resolving time will be different if the computing units numbers of previous GMRES and ERAM Components are different.

4.6 Scalability Evaluation

The main concern of preconditioned Krylov methods is the cost of per iteration, because of the global communication and synchronization overheads. In order to evaluate the performance of UCGLE method on both CPU and GPU clusters, we evaluate its strong scalability comparing with the classic and preconditioned GMRES by the average time cost per iteration. The test matrix is *MEG*1. The average time cost for these methods is computed by a fixed number of iterations. Time per iteration is suitable for demonstrating scaling behavior.

For the evaluation of UCGLE method on the homogeneous cluster, the four computing components are all implemented on the CPUs, the core number of GMRES Component is set respectively to be 1, 2, 4, 8, 16, 32, 64, 128, 256, and both the core number of LS Component and Manager Component is 1. ERAM Component should ensure to supply the approximated eigenvalues in time for each time restart of GMRES Component, thus the core number is respectively 1, 1, 1, 1, 4, 4, 4, 10, 16, referring to different GMRES Component core number. For the evaluation on the heterogeneous cluster, GMRES Component and ERAM Component are implemented on GPUs, both LS Component and Manager Component allocate only 1 core on CPU. The GPU number of GMRES Component is set respectively to be 2, 4, 8, 16, 32, 64, 128, with the GPU number of ERAM Component respectively 1, 1, 1, 4, 4, 4, 8. The computing resource number of classic and preconditioned GMRES keeps always the same with the core number of GMRES Component in UCGLE method, thus it ranges from 1 to 256 for CPU performance evaluation, and from 2 to 128 for GPU performance evaluation.

In the experimentations, firstly we can find that UCGLE method is able to take advantages of the GPU accelerators which has almost 4 times speed up for the time cost per iteration comparing with the homogenous cluster without accelerators.

In Figure 6, we can find that these methods have good scalability both on CPU and GPU with the augmentation of computing units except the SOR preconditioned GMRES. The classic GMRES has smallest time cost per iteration. The Jacobi preconditioner is the simplest preconditioning form for GMRES. This time cost gap between Jacobi preconditioned GMRES and classic GMRES is not enormous. The GMRES with SOR preconditioner has the largest time cost per iteration since SOR preconditioned GMRES has the additional matrix-vector and matrix-matrix multiplication operations in each step of the iteration. These operations have global

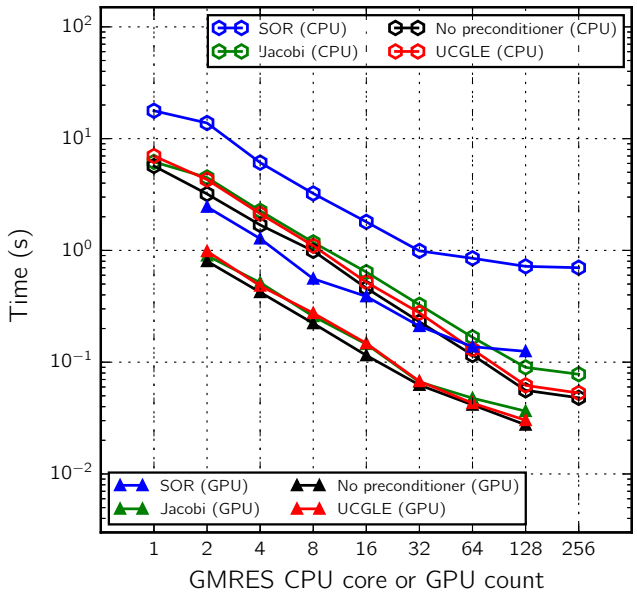

Figure 6: Strong scalability test of solve time per iteration for UCGLE, GMRES without preconditioner, Jacobi and SOR preconditioned GMRES using matrix *MEG*1 on CPU and GPU; X-axis refers respectively to CPU cores of GMRES from 1 to 256 and GPU number of GMRES from 2 to 128; Y-axis refers to the average execution time per iteration. A base 2 logarithmic scale is used for X-axis, and a base 10 logarithmic scale is used for Y-axis.

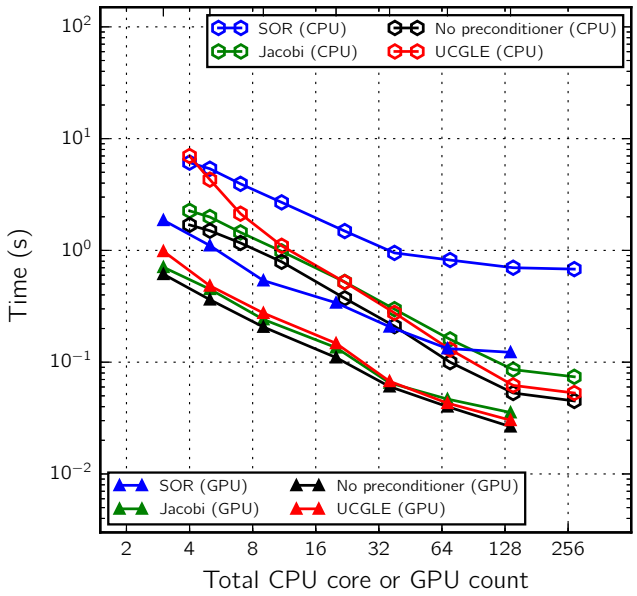

Figure 7: Performance comparison of solve time per iteration for UCGLE, GMRES without preconditioner, Jacobi and SOR preconditioned GMRES using matrix *MEG*1 on CPU and GPU; X-axis refers respectively to the total CPU cores number or GPU number for these four methods; Y-axis refers to the average execution time per iteration. A base 2 logarithmic scale is used for X-axis, and a base 10 logarithmic scale is used for Y-axis.

communication and synchronization points. The communication overhead makes the SOR preconditioned GMRES much more easily lose its good scalability with the augmentation of computing unit number. There isn't much difference between the time cost per iteration of classic GMRES and UCGLE. This phenomenon is caused by the asynchronous communication of UCGLE method. Since the resolving part and preconditioning part of UCGLE work independently, the global communication and synchronize points of UCGLE is similar with the classic GMRES for each time of preconditioning. That's the benefits of UCGLE and its asynchronous communication.

4.7 Performance Evaluation

The average time cost per iteration is also used to evaluate their performance. For the performance comparison, it is necessary to keep the total computing resource number of UCGLE and other methods the same. We have tested the classic and conventional preconditioned GMRES with the CPU core number fixed respectively as 4, 5, 7, 11, 22, 38, 70, 140, 274 and the GPU number fixed respectively as 3, 5, 9, 20, 36, 68, 136, referring to the previous evaluation of UCGLE method in Section 4.6. In the evaluation on GPU cluster, the two CPUs for LS Component and Manager Component have been ignored because they have a minor influence.

The performance comparison is given in Figure 7. We can find that if the computing resource number is small, the performance of classic and conventional preconditioned GMRES is much better than UCGLE since the latter allocates extra computing resources for other components. With the augmentation of computing resources, if the CPU core number is larger than 22 or the GPU number is larger than 5, it comes that the etra computing ressources have slight influence on the resolution, and the average time cost per iteration of UCGLE method trends to be similar to the ones of classic GMRES, and to be much better than the conventional preconditioned, especially the SOR preconditioned GMRES. For test matrix *MEG*1, UCGLE method has similar speedup on the solving time per iteration comparing with the classic GMRES when the computing resource number is larger than 22 for CPUs and larger than 5 for GPUs, but it can decrease significantly more than 5× iteration step number for the convergence, thus about 5× acceleration for the time of the whole resolution. In the end, the better performance of UCGLE method comparing with other methods can be concluded.

5 CONCLUSION AND PERSPECTIVES

In this paper, we have presented a distributed and parallel method UCGLE for solving large-scale non-Hermitian linear systems. This method has been implemented with asynchronous communication among different computation components. In the experimentation, we observed that UCGLE method has following features: 1) it has significant acceleration for the convergence than the conventional preconditioners as SOR and Jacobi; 2) the spectrum of different linear systems has influence on its improvement of convergence rate; 3) it has better scalability for the very large-scale linear systems;

4) it is able to speed up using GPUs; 5) it has the fault tolerance mechanism facing the failure of different computation components.

We conclude that UCGLE method is a good candidate for emerging large-scale computational systems because of its asynchronous communication scheme, its multi-level parallelism, its reusability and fault tolerance and its potential load balancing. The coarse grain parallelism among different computation components and the medium/fine grain parallelism inside each component can be flexibly mapped to large-scale distributed hierarchical platforms.

Various parameters have an impact on the convergence. Thus, an auto-tuning is required in the future work, where the systems can select different Krylov subspace dimensions, numbers of eigenvalues to be computed, degrees of Least Squares polynomial according to different linear systems and cluster architectures.

PETSc and SLEPc are not well compatible with the Intel Xeon Phi architectures. Additionally, other hybrid methods can be proposed with different computing components using unite and conquer approach, but it is difficult to implement them without the knowledge of MPI communication. YvetteML (YML) [7] is a workflow language based on user-friendly and hierarchical system-oriented development and execution environment, and XcalableMP (XMP) [14] is a directive-based language extension for distributed memory systems. With YML, the user can automatically allocate the computing units and establish the inter-communication for different tasks. It is necessary to develop a new version of UCGLE method with this multi-level languages YML-XMP for various architectures.

ACKNOWLEDGMENTS

The authors would like to thank ROMEO HPC Center-Champagne Ardenne for their support in provinding the use of cluster *ROMEO*. This work is supported by the project *MYX* of *French National Research Agency (ANR)* with the project ID: ANR-15-SPPE-003.

REFERENCES

[1] Tofigh Allahviranloo. 2005. Successive over relaxation iterative method for fuzzy system of linear equations. *Appl. Math. Comput.* 162, 1 (2005), 189–196.

[2] Walter Edwin Arnoldi. 1951. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics* 9, 1 (1951), 17–29.

[3] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. 2016. *PETSc Users Manual.* Technical Report ANL-95/11 - Revision 3.7. Argonne National Laboratory. http://www.mcs.anl.gov/petsc

[4] Claude Brezinski and M Redivo-Zaglia. 1994. Hybrid procedures for solving linear systems. *Numer. Math.* 67, 1 (1994), 1–19.

[5] SH Chan, KK Phoon, and FH Lee. 2001. A modified Jacobi preconditioner for solving ill-conditioned Biot's consolidation equations using symmetric quasi-minimal residual method. *International Journal for Numerical and Analytical Methods in Geomechanics* 25, 10 (2001), 1001–1025.

[6] Edmond Chow and Yousef Saad. 1997. Experimental study of ILU preconditioners for indefinite matrices. *J. Comput. Appl. Math.* 86, 2 (1997), 387–414.

[7] Olivier Delannoy, FranceNahid Emad, and Serge Petiton. 2006. Workflow global computing with yml. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*. IEEE Computer Society, 25–32.

[8] Jack Dongarra, Jeffrey Hittinger, John Bell, Luis Chacon, Robert Falgout, Michael Heroux, Paul Hovland, Esmond Ng, Clayton Webster, and Stefan Wild. 2014. *Applied mathematics research for exascale computing.* Technical Report. Lawrence Livermore National Laboratory (LLNL), Livermore, CA.

[9] Nahid Emad and Serge Petiton. 2016. Unite and conquer approach for high scale numerical computing. *Journal of Computational Science* 14 (2016), 5–14.

[10] Nahid Emad, Serge Petiton, and Guy Edjlali. 2005. Multiple explicitly restarted Arnoldi method for solving large eigenproblems. *SIAM Journal on Scientific Computing* 27, 1 (2005), 253–277.

[11] Jocelyne Erhel, Kevin Burrage, and Bert Pohl. 1996. Restarted GMRES preconditioned by deflation. *Journal of computational and applied mathematics* 69, 2 (1996), 303–318.

[12] Alexandre Fender, Nahid Emad, Serge Petiton, and Joe Eaton. 2016. Leveraging accelerators in the multiple implicitly restarted Arnoldi method with nested subspaces. In *Emerging Technologies and Innovative Business Practices for the Transformation of Societies (EmergiTech), IEEE International Conference on.* IEEE, 389–394.

[13] David S Kershaw. 1978. The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations. *J. Comput. Phys.* 26, 1 (1978), 43–65.

[14] Jinpil Lee and Mitsuhisa Sato. 2010. Implementation and performance evaluation of xcalablemp: A parallel programming language for distributed memory systems. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on.* IEEE, 413–420.

[15] Richard B Lehoucq and Danny C Sorensen. 1996. Deflation techniques for an implicitly restarted Arnoldi iteration. *SIAM J. Matrix Anal. Appl.* 17, 4 (1996), 789–821.

[16] Thomas A Manteuffel. 1977. The Tchebychev iteration for nonsymmetric linear systems. *Numer. Math.* 28, 3 (1977), 307–327.

[17] Youcef Saad. 1984. Chebyshev acceleration techniques for solving nonsymmetric eigenvalue problems. *Math. Comp.* 42, 166 (1984), 567–588.

[18] Youcef Saad. 1987. Least squares polynomials in the complex plane and their use for solving nonsymmetric linear systems. *SIAM J. Numer. Anal.* 24, 1 (1987), 155–169.

[19] Yousef Saad. 2011. *Numerical Methods for Large Eigenvalue Problems: Revised Edition.* SIAM.

[20] Youcef Saad and Martin H Schultz. 1986. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing* 7, 3 (1986), 856–869.

[21] Yousef Saad, Manshung Yeung, Jocelyne Erhel, and Frédéric Guyomarc'h. 2000. A deflated version of the conjugate gradient algorithm. *SIAM Journal on Scientific Computing* 21, 5 (2000), 1909–1926.

[22] Gerard LG Sleijpen and Diederik R Fokkema. 1993. BiCGstab (l) for linear equations involving unsymmetric matrices with complex spectrum. *Electronic Transactions on Numerical Analysis* 1, 11 (1993), 2000.